Network Fault Tolerance in LA-MPI

Rob T. Aulwes, David J. Daniel, Nehal N. Desai,
Richard L. Graham, L. Dean Risinger,
Mitchel W. Sukalski, and Mark A. Taylor [**]

Los Alamos National Laboratory, Advanced Computing Laboratory,
MS-B287, P. O. Box 1663, Los Alamos NM 87545, USA
lampi-support@lanl.gov
http://www.acl.lanl.gov/la-mpi/

Abstract. LA-MPI is a high-performance, network-fault-tolerant implementation of MPI designed for terascale clusters that are inherently unreliable due to their very large number of system components and to trade-offs between cost and performance. This paper reviews the architectural design of LA-MPI, focusing on our approach to guaranteeing data integrity. We discuss our network data path abstraction that makes LA-MPI highly portable, gives high-performance through message striping, and most importantly provides the basis for network fault tolerance. Finally we include some performance numbers for Quadrics Elan, Myrinet GM and UDP network data paths.

1 Introduction

LA-MPI [1, 2] is an implementation of the Message Passing Interface (MPI) [3, 4] motivated by a growing need for fault tolerance at the software level in large high-performance computing (HPC) systems.

This need is caused by the sheer number of components present in modern HPC systems, particularly clusters. The individual components – processors, memory modules, network interface cards (NICs), etc. – are typically manufactured to tolerances adequate for small or desktop systems. When aggregated into a large HPC system, however, system-wide error rates may be too great to successfully complete a long application run [5]. For example, a network device may have an error rate which is perfectly acceptable for a desktop system, but not in a cluster of thousands of nodes, which must run error free for many hours or even days to complete a scientific calculation.

LA-MPI has two primary goals: *network fault tolerance* and *high performance*. Fortunately these goals are partially complimentary, since the flexible approach we take to the use of redundant network data paths to support fault

[**] Los Alamos report LA-UR-03-2939. Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36. Project support was provided through ASCI/PSE and the Los Alamos Computer Science Institute.

tolerance also allows LA-MPI to exploit all the available network bandwidth in a network-device-rich system by sending different messages and/or message-fragments over multiple network paths.

A well-known solution to the network fault tolerance problem is to use the TCP/IP protocol. We believe, however, that this protocol – developed to handle unreliable, inhomogeneous and oversubscribed networks – performs poorly and is overly complex for HPC system messaging. Instead LA-MPI implements a more limited but highly efficient checksum/retransmission protocol, which we discuss in detail in section 3.

There are several other approaches to fault-tolerant message passing systems [6–9], but these have tended to focus on the issues of *process* fault tolerance and assume the existence of a perfectly reliable network (typically TCP/IP). We do intend to explore process fault tolerance in future, but believe that a high performance, network-fault-tolerant messaging system is a necessary first step.

Other important features of LA-MPI include an open source license, standards compliance (MPI version 1.2 integrated with ROMIO [10] for MPI-IO v2 support), thread safety, and portability to many operating systems, processor architectures and network devices.

In the following sections we will first review the architecture of LA-MPI emphasizing the important role of our network data path abstraction. A detailed discussion of LA-MPI's data integrity protocol is given in section 3, while a selection of performance results are presented in section 4.

2 Architecture

At a high level, LA-MPI's architecture falls into two layers: an upper MPI layer, implementing the full richness of the MPI standard, and a lower User Level Messaging (ULM) layer that provides a simpler reliable message passing API. Looking deeper, ULM is itself composed of two layers: a Memory and Message Layer (MML), and a Send and Receive Layer (SRL). The MML consists of code common to all systems and data paths, while the SRL is highly device-specific.

Before discussing these layers and their interaction in more detail, it is helpful to discuss what types of network fault tolerance are provided by LA-MPI. We distinguish two separate functionalities: (a) guaranteed data integrity of delivered messages; and (b) the ability to fail-over from one network device to another if the first is generating too many errors. Both of these require that we are able to treat each available network device on an equal footing, and this has led us to develop an abstraction called a *network data path object* or, more succinctly, a *path*.

A path is an abstraction of lower-level network transports and devices that are available to LA-MPI. Each path can represent a single network adapter, or a set of common adapters, or even a common protocol over many different network adapters. Currently, paths are implemented for shared memory, UDP (over all IP-enabled devices), Quadrics Elan3 [11, 12] remote direct memory access (RDMA) and Myrinet GM [13], and is currently being developed for Infiniband

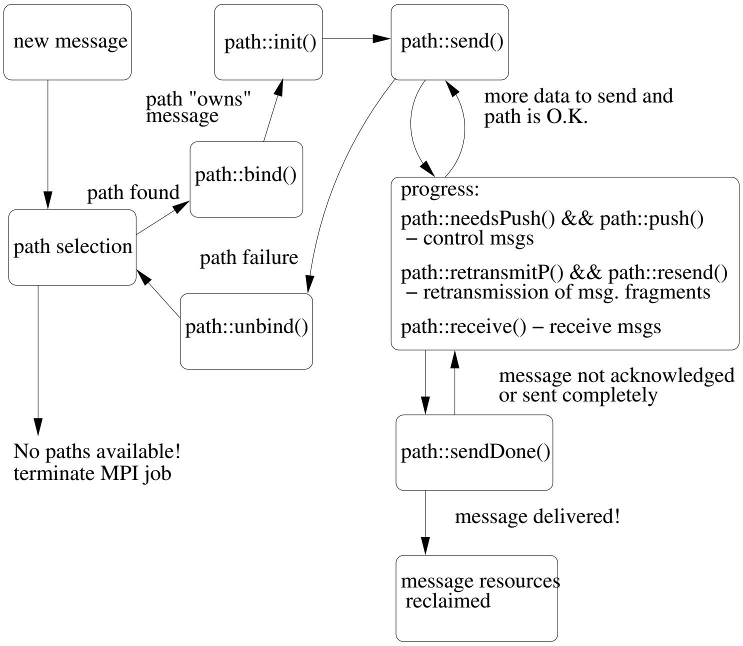
new message
path::init()
path::send()
path "owns" message
more data to send and path is O.K.
path::bind()
path found
progress:
path::needsPush() && path::push() – control msgs
path::retransmitP() && path::resend() – retransmission of msg. fragments
path::receive() – receive msgs
path selection
path failure
path::unbind()
message not acknowledged or sent completely
path::sendDone()
No paths available! terminate MPI job
message delivered!
message resources reclaimed

Fig. 1. Message-Path Interactions.

and TCP. In all of our current paths except UDP/IP, which treats multiple network adapters as a single Internet Protocol "device", multiple network adapters are used by a single path instantiation, if they exist on the machine.

The path object provides the interface between the portable common code of the MML and device-specific SRL. The interaction of the MML with the various paths is described schematically in figure 1.

As noted in the introduction, the path model enables us to implement message striping, where different messages may be sent along different data paths. For paths that comprise several NICs, we also stripe fragments of a single message across the NICs, and so achieve excellent network bandwidth.

An additional benefit of the path abstraction is that it enforces a high degree of portability in LA-MPI. For example, since differenct messages may be sent along different paths, all tag matching must be done in the MML in a path-independent way in order to respect MPI ordering semantics. We have found that there is no performance penalty for this approach (see section 4), while we gain in terms of an increasingly clean and maintainable code base.

In the next section we give an in-depth discussion of one aspect of network fault tolerance in LA-MPI, namely support for reliable message delivery, that is, the guaranteed integrity of delivered data. The other aspects of LA-MPI's architecture are described in more detail elsewhere [2].

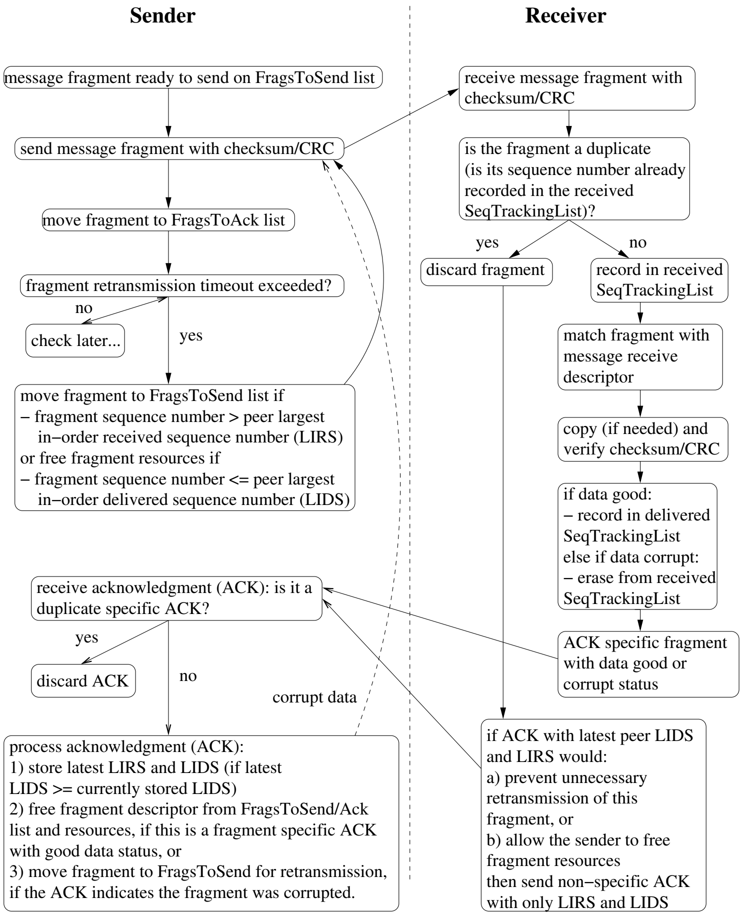
Sender
Receiver
message fragment ready to send on FragsToSend list
send message fragment with checksum/CRC
move fragment to FragsToAck list
fragment retransmission timeout exceeded?
no
check later...
yes
move fragment to FragsToSend list if
– fragment sequence number > peer largest in–order received sequence number (LIRS)
or free fragment resources if
– fragment sequence number <= peer largest in–order delivered sequence number (LIDS)
receive acknowledgment (ACK): is it a duplicate specific ACK?
yes
discard ACK
no
corrupt data
process acknowledgment (ACK):
1) store latest LIRS and LIDS (if latest LIDS >= currently stored LIDS)
2) free fragment descriptor from FragsToSend/Ack list and resources, if this is a fragment specific ACK with good data status, or
3) move fragment to FragsToSend for retransmission, if the ACK indicates the fragment was corrupted.
receive message fragment with checksum/CRC
is the fragment a duplicate (is its sequence number already recorded in the received SeqTrackingList)?
yes
no
discard fragment
record in received SeqTrackingList
match fragment with message receive descriptor
copy (if needed) and verify checksum/CRC
if data good:
– record in delivered SeqTrackingList
else if data corrupt:
– erase from received SeqTrackingList
ACK specific fragment with data good or corrupt status
if ACK with latest peer LIDS and LIRS would:
a) prevent unnecessary retransmission of this fragment, or
b) allow the sender to free fragment resources
then send non–specific ACK with only LIRS and LIDS

Fig. 2. Retransmission and Checksumming.

3 Reliability

Unlike many MPI libraries that consider all underlying communication perfectly reliable, LA-MPI optionally supports sender-side retransmission of messages by checking an “unacknowledged” list periodically for message send descriptors that have exceeded their timeout periods. This retransmission scheme is appropriate for low error rate environments, typical of most clusters. Each network transport is responsible for arranging to retransmit the necessary fragments. Each fragment’s retransmission time is calculated using a truncated exponential back-off scheme; this avoids resource exhaustion at a receiving process that is busy doing non-MPI computation. Fragments that must be retransmitted are moved from the `FragsToAck` list to the `FragsToSend` list, and the associated message send descriptor is placed on the incomplete list.

System	Path	Latency (μs)	Bandwidth (MB/s)
alpha	Shared Memory	2.93	935
alpha	Quadrics/Elan (1 NIC)	11.23 (8.39)	257 (273)
alpha	Quadrics/Elan (2 NICs)	11.37 (8.43)	438 (468)
alpha	Quadrics/Elan (UDP/IP)	156	67
i686	UDP/IP gigE	125.1	91
i686	Shared Memory	3.09	455
i686	Myrinet/GM (1 NIC)	11.91	241
i686	Myrinet/GM (2 NICs)	12.26	403
i686	Myrinet/GM (UDP/IP)	125.2	94
i686	UDP/IP gigE	125.1	91

Table 1. Zero-byte latency and peak point-to-point bandwidth for various LA-MPI paths. For the Quadrics path, we also give (in parentheses) the performance numbers with reliability (guaranteed data integrity) turned off (a run-time option).

Each network transport is also responsible for providing a main memory-to-main memory 32-bit additive checksum or 32-bit cyclic redundancy code (CRC), if it is needed. This checksum/CRC protects against network and I/O bus corruption, and is generated at the same time data is copied, if at all possible. By delaying checksumming to avoid wasting memory bandwidth, a received fragment is not necessarily a deliverable, or uncorrupted, fragment. The checksum can be disabled at run-time for additional performance at the cost of guaranteed data integrity.

Several MML generic features aid in the implementation of this retransmission and checksumming scheme. Every byte of data sent between a given pair of processes is associated with a monotonically increasing 64-bit sequence number. A fragment is therefore labeled by a range of sequence numbers (as a special case, zero-byte messages are assigned a single sequence number). The receiving process records the sequence numbers of arriving fragments in a special object, `SeqTrackingList`, as an ordered set of possibly non-contiguous ranges of sequence numbers. These lists use internal hint pointers to exploit any temporal locality in accessing these lists to minimize access overhead. The receiver maintains two `SeqTrackingList` lists for each peer with which it communicates to distinguish between fragments that have been received, and those that have been received and delivered successfully (i.e., no data corruption). Duplicate fragments are easily detected by checking the received fragment's sequence number range against the received `SeqTrackingList`.

We use per-byte rather than per-fragment sequence numbers to support network fault tolerance: by keeping track of the delivery of individual bytes we can more easily rebind failed fragment transmissions to alternate network paths with different fragment sizes. This is outlined in figure 1 and discussed more fully elsewhere [2].

Upon processing fragment acknowledgments from a receiver, a sender will store two special values that are carried in every acknowledgment: the largest

Implementation	Path	Latency (μs)	Bandwidth (MB/s)
LA-MPI	Shared Memory	2.93	935
LA-MPI	Quadrics/Elan (1 NIC)	11.23 (8.39)	257 (273)
LA-MPI	Quadrics/Elan (2 NICs)	11.37 (8.43)	438 (468)
MPICH	Quadrics/Elan (1 NIC)	4.65	228
MPICH	Quadrics/Elan (2 NICs)	4.57	257
HP/Compaq	Quadrics/Elan (1 NIC)	4.89	292
HP/Compaq	Quadrics/Elan (2 NICs)	4.99	258

Table 2. Zero-byte latency and peak point-to-point bandwidth for several MPI implementations (LA-MPI, MPICH and HP/Compaq MPI ("alaska"), on the alpha system described in table 1. The LA-MPI numbers in parentheses are those with reliability turned off (a run-time option).

in-order peer received sequence number (LIRS), and the largest in-order peer delivered sequence number (LIDS). The LIRS is used to prevent the retransmission of fragments that have been received, but whose data integrity has not been checked yet; it may increase or decrease over time, depending upon transmission and I/O bus errors. The LIDS is used to free any fragments whose acknowledgment was lost. The LIDS is always less than or equal to the LIRS. Figure 2 shows the interaction of these sequence numbers, the retransmission scheme, and checksumming.

4 Performance

In this section we present benchmark results that characterize the performance of LA-MPI on a variety of computer architectures, and allow a comparison with other implementations of MPI.

In table 1 we give "ping-pong" performance results from two systems currently of interest to us: (a) an alpha/Tru64 system consisting of HP/Compaq ES45 4-way nodes with 1.25 GHz alpha ev68 processors, and a "dual rail" Quadrics Elan/Elite interconnect; and (b) an i686/Linux system composed of 2 GHz dual Xeon nodes with 2 Myrinet 2000 cards.

Also included in table 1 are results for LA-MPI's UDP/IP path run over Elan/IP, and Myrinet GM/IP. These numbers give an indication of the very large cost associated with a complete IP implementation, and why LA-MPI uses the light-weight checksum/retransmission protocol described in section 3.

For the Quadrics path on alpha we quote the results with and without data integrity guaranteed. As can be seen the impact of reliability on performance is relatively small, increasing latency by about a third and reducing bandwidth by less than 10 %.

Table 2 gives a comparison of LA-MPI with two vendor supplied MPI implementations: MPICH as optimized by Quadrics for the Elan/Elite network, and HP/Compaq MPI version r11 (also known as "alaska"). Both of these implementations are based on the native Quadrics `libelan` tagged-messaging primitive elan_tport, whereas LA-MPI uses the common code in the MML for tag matching and accesses the Elan using libelan3 chained DMAs [11].

In fairness we should point out that the Quadrics native Elan library achieves a ping-pong latency of about 4.5 μs. There are several structural reasons for our higher latency (8.39), mainly related to the way that tag matching is implemented – in order to support all paths LA-MPI can make fewer assumptions about message-fragment arrival. We believe, however, that further refinements to LA-MPI's MML and SRL will reduce this gap.

For "on-host" messages, on the other hand, LA-MPI can use its shared memory path which easily out-performs the Elan network; this option is not available to the elan_tport-based approaches.

LA-MPI truly excels in the bandwidth benchmarks, for two reasons. Firstly, on-host traffic is handled by the shared memory path which has a much higher bandwidth than the Elan devices. Secondly, on systems with two "rails" of Elan/Elite network (i.e. two Elan devices per node), LA-MPI highly efficiently sends message fragments along both rails. In this simple ping-pong benchmark the elan_tport-based libraries show little or negative improvement with two rails, because they use the rails by assigning messages between a process pair to a fixed rail. For some communication patterns this approach may be reasonably efficient, but we emphasize that LA-MPI's fragment-based scheduling across the rails is efficient for all communication patterns.

5 Conclusions

With the rise of terascale distributed computing environments consisting of thousands of processors and network adapters, the need for fault tolerant software has become critical to their successful use. Negligible component error and failure rates in small to medium size clusters are no longer negligible in these large clusters, due to their complexity, sheer number of components, and amount of data transferred.

LA-MPI addresses the network-related challenges of this environment by providing a production-quality, reliable, high-performance MPI library for applications capable of (a) surviving network and I/O bus data corruption and loss, and (b) surviving network hardware and software failure if other connectivity is available.

Future development efforts will address (a) the implementation of a fault-tolerant, scalable, administrative network for job control, standard I/O redirection, and MPI wire-up; (b) the implementation of process fault-tolerance in the face of multiple process failures; and (c) the implementation of dynamic topology reconfiguration and addition of MPI processes to support dynamic process migration and MPI-2 dynamic processes.

LA-MPI is currently available as open source software under an LGPL license. It currently runs on Linux (i686 and Alpha processors), HP's Tru64 (Alpha only), SGI's IRIX 6.5 (MIPS), and Apple's Mac OS X (PowerPC). It supports an increasing variety of paths (network devices) as discussed in section 2. LA-MPI supports job spawning and control with Platform LSF, Quadrics RMS (Tru64 only), Bproc [14], and standard BSD `rsh`. Please send email to `lampi-support@lanl.gov`, and for more information visit our web site [15]. All fault tolerance features described in this paper have been fully implemented, except for on-going work on automatic network fail-over support.

References

1. Richard L. Graham, Sung-Eun Choi, David J. Daniel, Nehal N. Desai, Ronald G. Minnich, Craig E. Rasmussen, L. Dean Risinger, and Mitchel W. Sukalski. A network-failure-tolerant message-passing system for terascale clusters. In *Proceedings of the 16th international conference on Supercomputing*, pages 77–83. ACM Press, 2002.
2. Rob T. Aulwes, David J. Daniel, Nehal N. Desai, Richard L. Graham, L. Dean Risinger, and Mitchel W. Sukalski. LA-MPI: The design and implementation of a network-fault-tolerant MPI for terascale clusters. Technical Report LA-UR-03-0939, Los Alamos National Laboratory, 2003.
3. Message Passing Interface Forum. MPI: A Message Passing Interface Standard. Technical report, 1994.
4. Message Passing Interface Forum. MPI-2.0: Extensions to the Message-Passing Interface. Technical report, 1997.
5. C. Partridge, J. Hughes, and J. Stone. Performance of checksums and CRCs over real data. *Computer Communication Review*, v. 25 n. 4:68–76, 1995.
6. Georg Stellner. CoCheck: Checkpointing and Process Migration for MPI. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)*, Honolulu, Hawaii, 1996.
7. M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *8th International Conference on Distributed Computing System*, pages 108–111. IEEE Computer Society Press, 1988.
8. A. Agbaria and R. Friedman. Starfish: Fault-tolerant dynamic MPI programs on clusters of workstations. In *8th IEEE International Symposium on High Performance Distributed Computing*, 1999.
9. G. Fagg and a Dongarra. FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World. In *EuroPVM/ MPI User's Group Meeting 2000, Springer-Verlag, Berlin, Germany, 2000*, 2000.
10. Rajeev Thakur, William Gropp, and Ewing Lusk. *Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation.* Mathematics and Computer Science Division, Argonne National Laboratory, October 1997. ANL/MCS-TM-234.
11. Quadrics Ltd. *http://www.quadrics.com/*.
12. Fabrizio Petrini, Wu-Chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics network: High-performance clustering technology. *IEEE Micro*, v. 22 n. 1:46–57, 2002.
13. Myricom, Inc. *http://www.myri.com/*.
14. Cluster Research Team, Advanced Computing Laboratory, Los Alamos National Laboratory. *http://public.lanl.gov/cluster/index.html*.
15. Los Alamos Message Passing Interface, Advanced Computing Laboratory, Los Alamos National Laboratory. *http://www.ccs.lanl.gov/ccs1/projects/la-mpi*.